

BY KATIE SYLOR-MILLER
AND JULIA EVANS



♡

<https://dangitgit.com>
love this?



RECIPES FOR GITTING OUT OF A GIT MESS

♡

What's this?

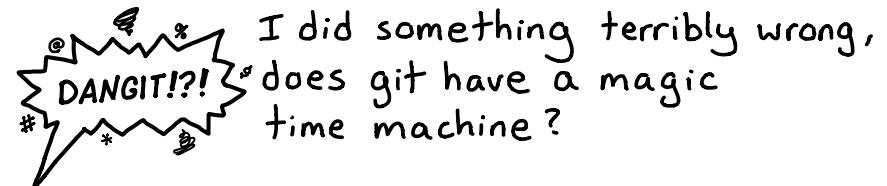
If you find git confusing, don't worry! You're not alone. People who've been using it every day for years still make mistakes and aren't sure how to fix them. A lot of git commands are confusingly named (why do you create new branches with `git checkout`?) and there are 20 million different ways to do everything.



This zine explains some git fundamentals in plain English, and how to fix a lot of common git mistakes.



By Katie Sylor-Miller & Julia Evans
Website: <https://ohshitgit.com>
Cover art by Deise Lino



Yes! It's called `git reflog` and it logs every single thing you do with git so that you can always go back.

Suppose you ran these git commands:

```
git checkout my-cool-branch ①  
git commit -am "add cool feature" ②  
git rebase master ③
```

Here's what `git reflog`'s output would look like. It shows the most recent actions first:

```
:  
245fc8d HEAD@{2} rebase -i (start):③ checkout master  
b623930 HEAD@{3} commit: ② add cool feature  
01d7933 HEAD@{4} checkout: ① moving from master  
to my-cool-branch:  
:
```

If you really regret that rebase and want to go back, here's how:

```
git reset --hard b623930  
git reset --hard HEAD@{3}
```

2 ways to refer to that commit before the rebase

..... 19
 I did something terribly wrong, does git have a
 magic time machine?

..... 18
 I want to undo something from 5 commits ago!

..... 17
 I want to split my commit into 2 commits!

..... 16
 I rebased and now I have 1,000 conflicts to fix!

..... 15
 I committed a file that should be ignored!

..... 14
 I have a merge conflict!

..... 13
 I tried to run a diff but nothing happened!

..... 12
 I accidentally committed to the wrong branch!

..... 10
 I committed but I need to make one small change!

..... 9
 I need to change the message on my last commit!

• dangit! mistakes & how to fix them •

..... 8
 mistakes you can't fix

..... 7
 HEAD is the commit you have checked out

..... 6
 a branch is a reference to a commit

..... 5
 every commit has a parent commit!

..... 4
 a SHA is always the same code

* git fundamentals *

Table of Contents

DANGIT!! I want to undo something from 5 commits ago!

If you made a mistake but want to keep all of the commits since then, git revert is your friend!

git revert will create a reverse patch for the changes in a commit and add it as a new commit.

① Find the commit SHA for the commit you

want to undo

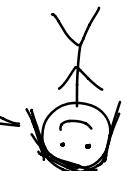
③ Enter a commit message for the revert commit

git revert SHA

② Run:

Now all of the changes you made in that commit are undone!

this is super useful if you push a bad commit to a shared repository and need to undo it!



A SHA always refers to the same code

Let's start with some fundamentals! If you understand the basics about how git works, it's WAY easier to fix mistakes. So let's explain what a git commit is!

Every git commit has an id like 3f29abcd233fa, also called a SHA ("Secure Hash Algorithm"). A SHA refers to both:

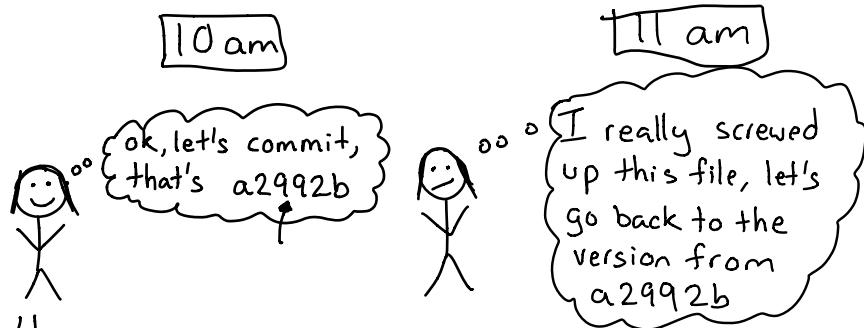
- the changes that were made in that commit see them with 'git show'
- a snapshot of the code after that commit was made

No matter how many weird things you do with git, checking out a SHA will always give you the exact same code. It's like saving your game so that you can go back if you die! You can check out a commit like this:

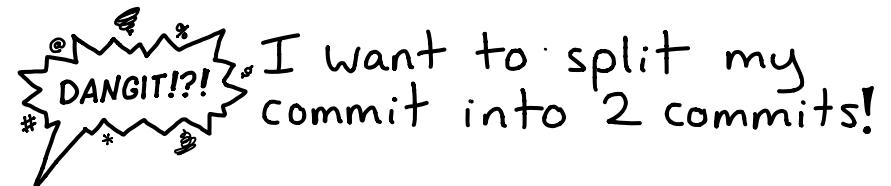
git checkout 3f29ab

SHAs are long
but you can
just use the
first 6 chars

This makes it way easier to recover from mistakes!



4



- ① Stash any uncommitted changes (so they don't get mixed up with the changes from the commit)

git stash

- ② Undo your most recent commit

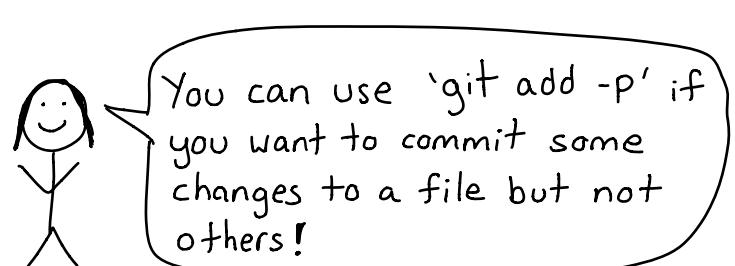
git reset HEAD^

↑
safe: this points your branch at the parent commit but doesn't change any files

- ③ Use git add to pick and choose which files you want to commit and make your new commits!

- ④ Get your uncommitted changes back

git stash pop



17

A branch is a pointer
to a commit

fix-fypa → qaqaqa

awesomenefeature → 3bafeaf

master → 2e9fab

A branch in git is a pointer to a commit SHA

Here's some proof! In your favourite git repo, run
this command:
git cat .git/refs/heads/master

this is just a file
with the commit SHA master
points at!

\$ cat .git/refs/heads/master

- * git commit will point the branch at the new commit
- * git push will point the branch at the same commit as the remote branch
- * git reset COMMIT-SHA will point the branch at COMMIT-SHA

3 main ways to change the commit a branch points to:

commit again!

Understand what a branch is will make it WAY EASIER to figure out how to get your branch to point at the right commit again!

DANGEROUSLY
git rebase -i SHA_YOU_FOUND

16
branches with many conflicts
differently, if you have 2
commits, you can just merge!

git rebase master
④ Rebase on master
git merge-base
outputs of
goes here

git rebase -i SHA_YOU_FOUND

③ Squash all the commits in your branch together

git merge-base master my-branch

② Find the commit where your branch diverged from master

git rebase --abort

① Escape the rebase of doom

This can happen when you're rebasing many commits at once.

I started rebasing and now I have 10000000 conflicts to fix!
DANGER!!

HEAD is the commit you have checked out

In git you always have some commit checked out. HEAD is a pointer to that commit and you'll see HEAD used a lot in this zine. Like a branch, HEAD is just a text file.

Run `cat git/HEAD` to see the current HEAD.

Here are a couple of examples of how to use HEAD:

show the diff for the current commit:

```
git show HEAD
```

UNDO UNDO UNDO UNDO: reset branch to 16 commits ago ↴

```
git reset --hard HEAD~16
```

HEAD~16 means
16 commits ago

show what's changed since 6 commits ago:

```
git diff HEAD~6
```

squash a bunch of commits together

```
git rebase -i HEAD~8
```

Rebasing a branch against itself 8 commits ago lets you squash commits together!
(use "fixup")



I committed a file that should be ignored!

Did you accidentally commit a 1.5GB file along with the files you actually wanted to commit? We've all done it.

① Remove the file from Git's index

```
git rm --cached FILENAME
```

This is safe: it won't delete the file

② Amend your last commit

```
git commit --amend
```

③ (optional) Edit your `.gitignore` so it doesn't happen again



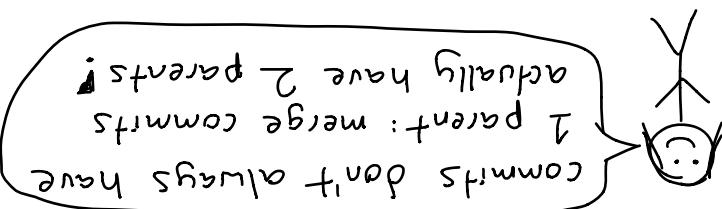
now your coworkers won't be stuck downloading a HUGE git commit

EVERY COMMIT (EXCEPT THE FIRST ONE!) HAS A PARENT COMMIT
 YOU CAN THINK OF YOUR GIT HISTORY AS LOOKING LIKE THIS:

```

    current          abcde HEAD "make cats blue"
    parent          feefef HEAD "add cats"
    grand-parent  a92eab HEAD "fix typo"
    :
    parent          b29aff HEAD "initial commit"
  
```

HEAD ALWAYS REFERS TO THE CURRENT COMMIT YOU HAVE
 CHECKED OUT, AND HEAD IS ITS PARENT. SO IF YOU WANT TO GO
 LOOK AT THE CODE FROM THE PREVIOUS COMMIT, YOU CAN RUN
 GIT CHECKOUT HEAD^



GIT CHECKOUT HEAD^

EVERY COMMIT HAS A PARENT

HELD (meldmerge.org) IS A GREAT CHOICE!
 RESOLVE CONFLICTS WITH 'GIT MERGE TOOL'.
 YOU CAN USE A GUI TO VISUALLY



- ① EDIT THE FILES TO FIX THE CONFLICT
- ② GIT ADD THE FIXED FILES
- ③ GIT DIFF --CHECK: CHECK FOR MORE CONFLICTS
- ④ GIT COMMIT WHEN YOU'RE DONE → OR GIT REBASE --CONTINUE IF YOU'RE REBASING!

TO RESOLVE THE CONFLICT:

```

<<<<< HEAD
  if x == 0:
    return false
  if y == 6:
    =====
    code from master
  else:
    return true
  if x == 0:
    feature - branch
    code from
  else:
    return false
>>>>> d34367
  
```

WHEN THAT CAUSES A MERGE CONFLICT, YOU'LL SEE SOMETHING LIKE THIS IN THE FILES WITH CONFLICTS:
 GIT MERGE FEATURE-BRANCH.

I HAVE A ?! MERGE CONFLICT?!
 DANGIT?!

mistakes you can't fix

Most mistakes you make with git can be fixed. If you've ever committed your code, you can get it back. That's what the rest of this zine is about!

Here are the dangerous git commands: the ones that throw away uncommitted work.



git reset --hard COMMIT

- ① Throws away uncommitted changes
- ② Points current branch at COMMIT

Very useful, but be careful to commit first if you don't want to lose your changes



git clean

Deletes files that aren't tracked by Git.



git checkout BRANCH FILE^{or directory}

Replaces FILE with the version from BRANCH.
Will overwrite uncommitted changes.



I tried to run a diff
but nothing happened?



did you know there are
3 ways to diff ??

Suppose you've edited 2 files:

\$ git status

On branch master

Changes to be committed:

modified: staged.txt

staged changes

(added with
'git add')

Changes not staged for commit:

modified: unstaged.txt

unstaged
changes

Here are the 3 ways git can show you a diff for these changes:

→ git diff: unstaged changes

→ git diff --staged: staged changes

→ git diff HEAD: staged+unstaged changes

A couple more diff tricks:

→ git diff --stat gives you a summary of which files were changed & number of added/deleted lines

→ git diff --check checks for merge conflict markers & whitespace errors

I need to change the message
on my last commit!
DANGER!!

No problem! Just run:

`git commit --amend`

Then edit the commit message & save!
git commit --amend will replace the old commit
with a new commit with a new SHA, so you can
always go back to the old version if you really need
to.

git checkout my-new-branch

① Make sure you have master checked out

git checkout master

② Create the new branch

④ Check out the new branch!

`git reset --hard HEAD~
git status`

③ Remove the unwanted commit from master

git branch my-new-branch

DANGER!! I committed something to
master that should have been
on a brand new branch!

'git branch', and 'git checkout -b', both
create a new branch. The difference is
'git checkout -b', also checks out the branch





I committed but I need to make one small change!

- ① Make your change
- ② Add your files with git add
- ③ Run:

```
git commit --amend --no-edit
```



this usually happens to me when I forget to run tests/ linters before committing!

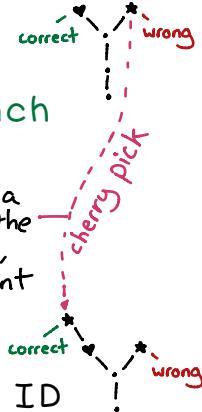
You can also add a new commit and use git rebase -i to squash them but this is about a million times faster.



I accidentally committed to the wrong branch!

- ① Check out the correct branch

```
git checkout correct-branch
```



cherry-pick makes a new commit with the same changes as *, but a different parent

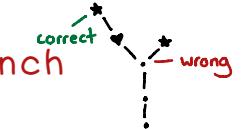
- ② Add the commit you wanted to it

```
git cherry-pick COMMIT_ID
```

use 'git log wrong-branch' to find this

- ③ Delete the commit from the wrong branch

```
git checkout wrong-branch  
git reset --hard HEAD^
```



be careful when running 'git reset --hard'! I always run 'git status' first to make sure there aren't uncommitted changes and 'git stash' to save them if there are